410 Rec'd PCT/PTO 10 SEP 2001

FORM-PTO-1390 (Rev. 12-29-99) U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE	ATTORNEY'S DOCKET NUMBER 032326-166
TRANSMITTAL LETTER TO THE UNITED STATES DESIGNATED/ELECTED OFFICE (DO/EO/US) CONCERNING A FILING UNDER 35 U.S.C. 371	U.S APPLICATION NO. (If known, see 37 C.F.R. 1.5) Unassigned 09/936174

INTERNATIONAL APPLICATION NO.	INTERNATIONAL FILING DATE	PRIORITY DATE CLAIMED
PCT/FR00/00150	January 24, 2000	9 March 1999

TITLE OF INVENTION
METHOD FOR MONITORING A PROGRAMME FLOW

APPLICANT(S) FOR DO/EO/US
GIRARD Pierre, ROUSSEAU Ludovic, and NACCACHE David

Applicant herewith submits to the United States Designated/Elected Office (DO/EO/US) the following items and other information:

1. ☒ This is a **FIRST** submission of items concerning a filing under 35 U.S.C. 371.
2. ☐ This is a **SECOND** or **SUBSEQUENT** submission of items concerning a filing under 35 U.S.C. 371.
3. ☒ This is an express request to begin national examination procedures (35 U.S.C. 371(f)) at any time rather than delay examination until the expiration of the applicable time limit set in 35 U.S.C. 371(b) and the PCT Articles 22 and 39(1).
4. ☒ A proper Demand for International Preliminary Examination was made by the 19th month from the earliest claimed priority date.
5. ☒ A copy of the International Application as filed (35 U.S.C. 371(c)(2))
   a. ☐ is transmitted herewith (required only if not transmitted by the International Bureau).
   b. ☒ has been transmitted by the International Bureau.
   c. ☐ is not required, as the application was filed in the United States Receiving Office (RO/US)
6. ☒ A translation of the International Application into English (35 U.S.C. 371(c)(2)).
7. ☒ Amendments to the claims of the International Application under PCT Article 19 (35 U.S.C. 371(c)(3))
   a. ☐ are transmitted herewith (required only if not transmitted by the International Bureau).
   b. ☐ have been transmitted by the International Bureau.
   c. ☐ have not been made; however, the time limit for making such amendments has NOT expired.
   d. ☒ have not been made and will not be made.
8. ☒ A translation of the amendments to the claims under PCT Article 19 (35 U.S.C. 371(c)(3)).
9. ☒ An oath or declaration of the inventor(s) (35 U.S.C. 371(c)(4)).
10. ☒ A translation of the annexes to the International Preliminary Examination Report under PCT Article 36 (35 U.S.C. 371(c)(5)).

Items 11. to 16. below concern other document(s) or information included:

11. ☒ An Information Disclosure Statement under 37 CFR 1.97 and 1.98.
12. ☐ An assignment document for recording. A separate cover sheet in compliance with 37 CFR 3.28 and 3.31 is included.
13. ☒ A FIRST preliminary amendment.
    ☐ A SECOND or SUBSEQUENT preliminary amendment.
14. ☐ A substitute specification.
15. ☐ A change of power of attorney and/or address letter.
16. ☐ Other items or information:

533 Rec'd PCT/PTO 10 SEP 2001

U.S. APPLICATION NO. (If known, see 37 CFR 1.50)	INTERNATIONAL APPLICATION NO	ATTORNEY'S DOCKET NUMBER
Unassigned 09/936174	PCT/FR00/00454	032326-163

				CALCULATIONS	PTO USE ONLY
17. ☒ The following fees are submitted:					
Basic National Fee (37 CFR 1.492(a)(1)-(5)): Neither international preliminary examination fee (37 CFR 1.482) nor international search fee (37 CFR 1.445(a)(2)) paid to USPTO and International Search Report not prepared by the EPO or JPO $1,000.00 (960)					
International preliminary examination fee (37 CFR 1.482) not paid to USPTO but International Search Report prepared by the EPO or JPO $860.00 (970)					
International preliminary examination fee (37 CFR 1.482) not paid to USPTO but international search fee (37 CFR 1.445(a)(2)) paid to USPTO $710.00 (958)					
International preliminary examination fee paid to USPTO (37 CFR 1.482) but all claims did not satisfy provisions of PCT Article 33(1)-(4) $690.00 (956)					
International preliminary examination fee paid to USPTO (37 CFR 1.482) and all claims satisfied provisions of PCT Article 33(1)-(4) $100.00 (962)					
ENTER APPROPRIATE BASIC FEE AMOUNT =				$ 860.00	
Surcharge of **$130.00 (154)** for furnishing the oath or declaration later than 20 ☐ 30 ☐ months from the earliest claimed priority date (37 CFR 1.492(e)).				$	
Claims	Number Filed	Number Extra	Rate		
Total Claims	30 -20 =	10	X$18.00 (966)	$ 180.00	
Independent Claims	3 -3 =		X$80.00 (964)	$	
Multiple dependent claim(s) (if applicable)			+$270.00 (968)	$	
TOTAL OF ABOVE CALCULATIONS =				$ 1,040.00	
Reduction for 1/2 for filing by small entity, if applicable (see below).				$	-
SUBTOTAL =				$	
Processing fee of **$130.00 (156)** for furnishing the English translation later than 20 ☐ 30 ☐ months from the earliest claimed priority date (37 CFR 1.492(f)). +				$	
TOTAL NATIONAL FEE =				$	
Fee for recording the enclosed assignment (37 CFR 1.21(h)). The assignment must be accompanied by an appropriate cover sheet (37 CFR 3.28, 3.31). **$40.00 (581)** per property +				$	
TOTAL FEES ENCLOSED =				$ 1,040.00	
				Amount to be: refunded	$
				charged	$

a. ☐ Small entity status is hereby claimed.

b. ☒ A check in the amount of $ 1,040.00 to cover the above fees is enclosed.

c. ☐ Please charge my Deposit Account No. 02-4800 in the amount of $_______ to cover the above fees. A duplicate copy of this sheet is enclosed.

d. ☒ The Commissioner is hereby authorized to charge any additional fees which may be required, or credit any overpayment to Deposit Account No. 02-4800. A duplicate copy of this sheet is enclosed.

NOTE: Where an appropriate time limit under 37 CFR 1.494 or 1.495 has not been met, a petition to revive (37 CFR 1.137(a) or (b)) must be filed and granted to restore the application to pending status.

SEND ALL CORRESPONDENCE TO:

James A. LaBarre
BURNS, DOANE, SWECKER & MATHIS, L.L.P.
P.O. Box 1404
Alexandria, Virginia 22313-1404
(703) 836-6620

James LaBarre
SIGNATURE

James A. LaBarre
NAME

28,632
REGISTRATION NUMBER

(01/01)

09/936174
533 Rec'd PCT/PTO 10 SEP 2001

Patent
Attorney's Docket No. 032326-166

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of	
GIRARD Pierre et al.	Group Art Unit: Unassigned
Application No.: Unassigned	Examiner: Unassigned
Filed: September 10, 2001	
For: METHOD FOR MONITORING A PROGRAMME FLOW	

09936174.091001

PRELIMINARY AMENDMENT

Assistant Commissioner for Patents
Washington, D.C. 20231

Sir:

Prior to examination and the calculation of filing fees, kindly amend the above-identified application as follows:

IN THE SPECIFICATION:

Page 1, immediately following the title appearing on line 1, insert the following:

--This disclosure is based upon, and claims priority from, French Application No. 99/02924 filed March 9, 1999 and International Application No. PCT/FR00/00150, which was published in a language other than English on September 14, 2000, the contents of each of which are incorporated herein by reference.

Background of the Invention--.

Page 6, between lines 26 and 27, insert the following heading:

--**Summary of the Invention**--.

Page 17, before line 5, insert the following heading:

--**Brief Description of the Drawings**--.

Page 18, before line 3, insert the following heading:

--**Detailed Description**--.

Add the following Abstract:

--In a method for monitoring the flow of execution of a series of instructions of a computer program, a sequence of instructions are transmitted to the processor to execute the monitored program. These instructions are analyzed, and the result of the analysis are verified by referring to reference data recorded with the program. The reference data can include a value predetermined in such a way as to correspond to the result of the analysis produced during the monitoring process only if all the instructions have been actually analyzed during the program flow. The invention also concerns a device for monitoring the program execution, a program device and a programming device operating according to the monitoring principles.--

IN THE CLAIMS:

Cancel claims 27 and 32.

Kindly replace claims 1-26 and 28-31, as follows.

1. (Amended) A method for monitoring progress with the execution of a linear sequence of instructions in a computer program, comprising the steps of analysing the sequence of instructions transmitted to a processor intended to execute the program being monitored by extracting a data item from each instruction transmitted to the processor and performing a calculation on said data item, and verifying the result of this analysis by comparing the result of said calculation to reference data, recorded with said program, wherein the reference data comprises a value pre-established so as to correspond to the result of the analysis produced during the monitoring method only if all the instructions in the sequence of instructions have actually been analysed during the running of the program.

2. (Amended) A method according to Claim 1, wherein the verification of the result of the analysis is caused by an instruction placed at a predetermined location in the program to be monitored, said instruction containing the reference data relating to a set of instructions whose correct execution is to be monitored.

3. (Amended) A method according to Claim 1 wherein, when the instructions of the set of instructions to be monitored are in the form of a value, said analysis of the instructions is carried out by using these instructions as a numerical value.

4. (Amended) A method according to Claim 1, comprising the steps of:

- during the preparation of the program to be monitored:

- incorporating, in at least one predetermined location in a sequence of instructions in the program, a reference value established according to a predetermined rule applied to identifiable data in each instruction to be monitored, and

- during the execution of the program to be monitored:

- obtaining said identifiable data in each instruction received for execution,

- applying said predetermined rule to said identifiable data thus obtained in order to establish a verification value, and

- verifying that this verification value actually corresponds to the reference value recorded with the program.

5. (Amended) A method according to Claim 1, further comprising a step of interrupting the flow of the program if the analysis reveals that the program being monitored has not been run as expected.

6. (Amended) A method according to Claim 1, further comprising an invalidation step for future use of the device comprising the monitored program if said analysis reveals a predetermined number of times that the program being monitored has not run in the expected manner.

7. (Amended) A method according to Claim 1 wherein the set of instructions to be monitored does not include jumps in its expected flow.

8. (Amended) A method according to Claim 1 wherein, when the program to be monitored provides for at least one jump, the monitoring method is applied separately to sets of instructions in the program which do not include jumps between two successive instructions.

9. (Amended) A method according to Claim 8, wherein, when the program to be monitored includes an instruction for a jump dependent on the manipulated data, the monitoring method is implemented separately for a set of instructions which precedes the jump, and for at least one set of instructions which follows said jump.

10. (Amended) A method according to Claim 9, wherein, for a set of instructions providing for a jump, an instruction which controls this jump is integrated in said set of instructions for the purpose of obtaining a verification value for this set of instructions before executing the jump instruction.

11. (Amended) A method according to Claim 1 wherein the analysis is reinitialised before each new monitoring of a sequence of instructions to be monitored.

12. (Amended) A method according to Claim 11, wherein the reinitialisation of the analysis of each new monitoring includes the step of erasing or replacing a verification value obtained during a previous analysis.

13. (Amended) A method according to Claim 11 wherein the reinitialisation of the monitoring analysis is controlled by the software itself.

14. (Amended) A method according to Claim 1 wherein the analysis produces a verification value obtained as the last value in a series of values which is made to change successively with the analysis of each of the analysed instructions of the set of instructions, thus making it possible to contain an internal state of the running of the monitoring method and to follow its changes.

15. (Amended) A method according to Claim 1 wherein the analysis includes the step of calculating, for each instruction under consideration following a previous instruction, the result of an operation on both a value obtained of the instruction in question and the result obtained by the same operation performed on the previous instruction.

16. (Amended) A method according to Claim 1 wherein the analysis includes the step of recursively applying a hash function to values obtained of each monitored instruction, starting from a last initialisation performed.

17. (Amended) A method according to Claim 1 wherein the analysis includes the step of making a verification value change by performing a redundancy calculation on all the operating codes and the addresses executed since the last initialisation was carried out.

18. (Amended) A method according to Claim 1 wherein the analysis includes the step of obtaining a comparison value by calculating successive intermediate values as the data of the respective instructions are obtained.

19. (Amended) A method according to Claim 1 wherein the analysis comprises a step of saving each data item necessary for verification, obtained from instructions in the set of instructions to be monitored as they are executed, and performing a calculation of a verification value from these data only at the necessary time, once all the necessary data have been obtained.

20. (Amended) A device for monitoring progress with the execution of a series of instructions of a computer program, comprising means for analysing the sequence of instructions transmitted to the processor intended to execute the program being monitored by extracting a data item from each instruction transmitted to the processor and performing a calculation on said data item, and means for verifying the result of this analysis by comparing the result of said calculation to reference data recorded with said program, wherein the reference data comprises a value pre-established so as to correspond to the

result of the analysis produced during monitoring only if all the instructions in the sequence of instructions have actually been analysed during the running of the program.

21. (Amended) A device according to Claim 20, further including a register for recording intermediate results in a calculation in a chain carried out by the analysis means in order to obtain a verification value.

22. (Amended) A device according to Claim 21, further comprising means for recording a predetermined value or resetting the register under the control of an instruction transmitted during the execution of a program to be monitored.

23. (Amended) A device according to Claim 20, further comprising means for counting the number of unexpected events in the program being monitored, as determined by the analysis means, and means for invalidating the future use of the program to be monitored if this number reaches a predetermined threshold.

24. (Amended) A device according to Claim 20 that is integrated into a programmed device containing said program to be monitored.

25. (Amended) A device according to Claim 20 that is integrated into a program execution device.

26. (Amended) A program execution device that executes a series of instructions of a computer program, comprising means for analysing the sequence of instructions transmitted for execution by extracting a data item from each instruction and performing a calculation on said data item, and means for verifying the result of this analysis by comparing the result of said calculation to reference data recorded with the program to be monitored, wherein the reference data comprises a value pre-established so as to correspond to the result of the analysis produced during monitoring only if all the instructions in the sequence of instructions have actually been analysed during the running of the program.

28. (Amended) A programmed device containing a series of recorded instructions and a fixed memory containing reference data pre-established as a function of data contained in said instructions for analysis and verification of the sequence of instructions, wherein the reference data comprises a value pre-established so as to correspond to the result of the analysis produced during monitoring only if all the instructions in the sequence of instructions have actually been analyzed during the running of the program.

29. (Amended) A device according to Claim 28, wherein said device is a smart card.

30. (Amended) A device according to Claim 28 wherein the reference data are recorded in the form of a prewired value or values fixed in memory.

31. (Amended) A device for programming a programmed device according to Claim 28, comprising means for entering, in at least one predetermined location in a sequence of instructions in the program, a reference value calculated according to a pre-established mode from data included in each instruction in a set of instructions whose execution is to be monitored.

REMARKS

Entry of the foregoing amendments is respectfully requested. This amendments are intended to place the claims in a more conventional format and eliminate the multiple dependency of the claims.

Respectfully submitted,

BURNS, DOANE, SWECKER & MATHIS, L.L.P.

By: [illegible]
James A. LaBarre
Registration No. 28,632

P.O. Box 1404
Alexandria, Virginia 22313-1404
(703) 836-6620

Date: September 10, 2001

Attachment to Preliminary Amendment dated September 10, 2001

Marked-up Claims 1-26 and 28-31

1. (Amended) A method for monitoring progress with the execution of a linear [series] sequence of instructions [(Inst.1-Inst.n)] in a computer program, [consisting in] comprising the steps of analysing the sequence of instructions transmitted to [the] a processor [(4)] intended to execute the program being monitored by extracting a data item from each instruction transmitted to the processor and performing a calculation on said data item, and verifying [and to verify] the result of this analysis by [reference] comparing the result of said calculation to reference data, [(Vref)] recorded with [the] said program, [characterised in that] wherein the reference data comprises a value [(Vref)] pre-established so as to correspond to the result of [this] the analysis produced during the monitoring method only if all the instructions [(Inst.1-Inst.n)] in the sequence of instructions have actually been analysed during the running of the program [and in that the said analysis of the sequence of instructions (Inst.1-Inst.n) comprises the extraction (38) of a data item from each instruction transmitted to the processor (4) and a predetermined calculation (40, 42) on each data item thus extracted, and in that the verification comprises a comparison (50) of the result of the analysis with the reference data (Vref)].

2. (Amended) A method according to Claim 1, [characterised in that the said] wherein the verification of the result of the analysis is caused by an instruction [(Inst.n+1)] placed at a predetermined location in the program to be monitored, [this] said instruction

Attachment to Preliminary Amendment dated September 10, 2001

Marked-up Claims 1-26 and 28-31

containing the reference data [(Vref)] relating to a set of instructions [(Inst.1-Inst.n)] whose correct execution is to be monitored.

3. (Amended) A method according to [any one of Claims 1 to 2, characterised in that] Claim 1 wherein, when the instructions [(Inst.1-Inst.n)] of the set of instructions to be monitored are in the form of a value, [for example codes recorded in hexadecimal or decimal form, the] said analysis of the instructions is carried out [considering] by using these instructions as a numerical value.

4. (Amended) A method according to Claim 1, comprising the steps [consisting in] of:

- during the preparation of the program to be monitored:

  - incorporating, [at] in at least one predetermined location in a sequence of instructions [(Inst.1-Inst.n)] in the program, a reference value [(Vref)] established according to a predetermined rule applied to identifiable data in each instruction to be monitored, and

- during the execution of the program to be monitored:

  - obtaining [(38) the] said identifiable data in each instruction received [with a view to its] for execution,

  - applying [(40, 42) the] said predetermined rule to [the] said identifiable data thus obtained in order to establish a verification value [(VHn)], and

Attachment to Preliminary Amendment dated September 10, 2001

Marked-up Claims 1-26 and 28-31

- verifying [(50)] that this verification value actually corresponds to the reference value recorded with the program.

5. (Amended) A method according to [any one of Claims 1 to 4, characterised in that it also comprises] Claim 1, further comprising a step [(56)] of interrupting the flow of the program [monitored] if the analysis reveals that the program being monitored has not been run as expected.

6. (Amended) A method according to [any one of Claims 1 to 5, characterised in that it also comprises] Claim 1, further comprising an invalidation step [(70)] for future use of the device comprising the monitored program if [the] said analysis reveals a predetermined number of times that the program being monitored has not run in the expected manner.

7. (Amended) A method according to [any one of Claims 1 to 6, characterised in that] Claim 1 wherein the set of instructions to be monitored does not include jumps in its expected flow.

8. (Amended) A method according to [any one of Claims 1 to 6, characterised in that] Claim 1 wherein, when the program [(EI1, EI2, EI3) or the program portion] to be

Attachment to Preliminary Amendment dated September 10, 2001

Marked-up Claims 1-26 and 28-31

monitored provides for at least one jump, the monitoring method is applied separately to sets of instructions in [this] the program which do not include jumps between two successive instructions.

9. (Amended) A method according to Claim 8, [characterised in that] wherein, when the program to be monitored includes an instruction [(EI1-j) giving rise to] for a jump dependent on the manipulated data, the monitoring method is implemented separately for a set of instructions [(EI1)] which precedes the jump, and for at least one set of instructions [(EI2, EI3)] which follows [this] said jump.

10. (Amended) A method according to Claim 9, [characterised in that] wherein, for a set of instructions [(EI1)] providing for a jump, [there is integrated in this set the] an instruction [(EI1-j)] which controls this jump is integrated in said set of instructions for the purpose of [the analysis aimed at] obtaining [the] a verification value [(VH)] for this set of instructions[, and thus the correct running of this set of instructions is verified] before executing the jump instruction.

11. (Amended) A method according to [any one of Claims 1 to 10, characterised in that] Claim 1 wherein the analysis is reinitialised before each new monitoring of a sequence [or set (EI1, EI2, EI3)] of instructions to be monitored.

Attachment to Preliminary Amendment dated September 10, 2001

Marked-up Claims 1-26 and 28-31

12. (Amended) A method according to Claim 11, [characterised in that] wherein the reinitialisation of the analysis of each new monitoring [consists in] includes the step of erasing or replacing a verification value [(VH)] obtained during a previous analysis.

13. (Amended) A method according to Claim 11 [or 12, characterised in that] wherein the reinitialisation of the monitoring analysis is controlled by the [protected] software itself.

14. (Amended) A method according to [any one of Claims 1 to 13, characterised in that] Claim 1 wherein the analysis produces a verification value [(VH)] obtained as the last value in a series of values which is made to change successively with the analysis of each of the analysed instructions [(Inst.1-Inst.n)] of the set of instructions, thus making it possible to contain an internal state of the running of the monitoring method and to follow its changes.

15. (Amended) A method according to [any one of Claims 1 to 14, characterised in that] Claim 1 wherein the analysis [consists in] includes the step of calculating [(40, 42)], for each instruction under consideration [(Inst.n)] following [on from] a previous instruction [(Inst.n-1)], the result of an operation on both a value [(VHn)]

Attachment to Preliminary Amendment dated September 10, 2001

Marked-up Claims 1-26 and 28-31

obtained of the instruction in question and the result [(VHn-1)] obtained by the same operation performed on the previous instruction.

16. (Amended) A method according to [any one of Claims 1 to 15, characterised in that] Claim 1 wherein the analysis [consists in] includes the step of recursively applying a hash function [f(VHn-1, Vinst.n)] to values obtained of each monitored instruction, starting from a last initialisation performed.

17. (Amended) A method according to [any one of Claims 1 to 15, characterised in that] Claim 1 wherein the analysis [consists in] includes the step of making a verification value change by performing a redundancy calculation[, not necessarily cryptographic,] on all the operating codes and the addresses executed since the last initialisation was carried out.

18. (Amended) A method according to [any one of Claims 1 to 17, characterised in that] Claim 1 wherein the analysis [consists in] includes the step of obtaining a comparison value [(VCn)] by calculating successive intermediate values as the data of the respective instructions [serving for this calculation during the execution of these instructions] are obtained.

Attachment to Preliminary Amendment dated September 10, 2001

Marked-up Claims 1-26 and 28-31

19. (Amended) A method according to [any one of Claims 1 to 17, characterised in that] Claim 1 wherein the analysis comprises a step of saving each data item necessary for verification, obtained from instructions in the set of instructions to be monitored [(Inst.1-Inst.n)] as they are executed, and performing a calculation of [the] a verification value [(VHn)] from these data only at the necessary time, once all the necessary data have been obtained.

20. (Amended) A device for monitoring progress with the execution of a series of instructions [(Inst.1-Inst.n)] of a computer program [implementing the monitoring method according to any one of Claims 1 to 19, characterised in that it comprises] , comprising means [(22-26)] for analysing the sequence of instructions transmitted to the processor [(4)] intended to execute the program being monitored by extracting a data item from each instruction transmitted to the processor and performing a calculation on said data item, and means [(26)] for verifying the result [(VCn)] of this analysis by [reference] comparing the result of said calculation to reference data [(Vref)] recorded with [the] said program, wherein the reference data comprises a value pre-established so as to correspond to the result of the analysis produced during monitoring only if all the instructions in the sequence of instructions have actually been analysed during the running of the program.

Attachment to Preliminary Amendment dated September 10, 2001

Marked-up Claims 1-26 and 28-31

21. (Amended) A device according to Claim 20, [adapted to implement the monitoring method according to any one of Claims 1 to 19, characterised in that it has] further including a register [(24)] for recording intermediate results [(VH)] in a calculation in a chain carried out by the analysis means [(26)] in order to obtain a verification value [(VHn)].

22. (Amended) A device according to Claim 21, [characterised in that it comprises] further comprising means for [allowing the] recording [of] a predetermined value or [a] resetting [of] the register [(24)] under the control of an instruction [(Inst.n+1)] transmitted during the execution of a program to be monitored[, for example on the occasion of a jump in the program].

23. (Amended) A device according to [any one of Claims 20 to 22, characterised in that it has a] Claim 20, further comprising means [(60)] for counting the number of unexpected events in the program being monitored, as determined by the analysis means [(26)], and means for invalidating the future use of the program to be monitored if this number reaches a predetermined threshold [(VCthreshold)].

Attachment to Preliminary Amendment dated September 10, 2001

Marked-up Claims 1-26 and 28-31

24. (Amended) A device according to [any one of Claims 20 to 23, characterised in that it] Claim 20 that is integrated into a programmed device[, such as a smart card,] containing [the] said program to be monitored.

25. (Amended) A device according to [any one of Claims 20 to 23, characterised in that it] Claim 20 that is integrated into a program execution device [(20)].

26. (Amended) A program execution device [(20) intended to execute] that executes a series of instructions [(Inst.1-Inst.n)] of a computer program, [characterised in that it has] comprising means [(22-26)] for analysing the sequence of instructions transmitted for execution by extracting a data item from each instruction and performing a calculation on said data item, and means for verifying the result of this analysis by [reference] comparing the result of said calculation to reference data [(Vref)] recorded with the program to be monitored, [according to any one of Claims 20 to 23] wherein the reference data comprises a value pre-established so as to correspond to the result of the analysis produced during monitoring only if all the instructions in the sequence of instructions have actually been analysed during the running of the program.

Attachment to Preliminary Amendment dated September 10, 2001

Marked-up Claims 1-26 and 28-31

28. (Amended) A programmed device containing a series of recorded instructions [(Inst.1-Inst.n), characterised in that it also contains] and a fixed memory containing reference data [(Vref)] pre-established as a function of data contained in [the] said instructions [and intended to allow] for analysis and verification of the sequence of instructions [analysed according to any one of Claims 1 to 21, the said device being intended to cooperate with a monitoring device according to any one of Claims 20 to 27], wherein the reference data comprises a value pre-established so as to correspond to the result of the analysis produced during monitoring only if all the instructions in the sequence of instructions have actually been analysed during the running of the program.

29. (Amended) A device according to Claim 28, [characterised in that it is in the form of] wherein said device is a smart card.

30. (Amended) A device according to Claim [30 or 31, characterised in that] 28 wherein the reference data [(Vref)] are recorded in the form of a prewired value or values fixed in memory.

31. (Amended) A device for programming a [device intended to be] programmed device according to [any one of Claims 28 to 30, characterised in that it comprises] Claim 28, comprising means for entering, [at] in at least one predetermined

Attachment to Preliminary Amendment dated September 10, 2001

Marked-up Claims 1-26 and 28-31

location in a sequence of instructions in the program [(Inst.1-Inst.n)], a reference value [(Vref)] calculated according to a pre-established mode from data included in each instruction in a set of instructions whose execution [it is wished to monitor] is to be monitored.

5/PRTS

METHOD FOR MONITORING A PROGRAM FLOW

The present invention relates to the field of computer program security, and more particularly a method and device for detecting unacceptable steps in the execution of a computer program, the latter being able to be in a low-level or high-level language.

In a program in low-level language, the commands are formulated according to a structure very close to that of the instructions actually executed by the processor part of the computer. The program requires only to be compiled before being able to be executed. Low-level languages, known by the name of machine code, are used notably for programming microprocessors or microcontrollers. Microcontrollers are processors which can execute only a small number of specific instructions. They are used notably for equipping smart cards (bank cards, telephone cards, cards providing access to services, etc) and for controlling industrial or domestic equipment.

In a program in high-level language, the commands have a structure closer to natural language, but on the other hand further away from that used by the processor. The commands written in such languages must first of all be interpreted, that is to say converted into machine code commands, before then being able to be put in the form of instructions with a view to their execution by the processor.

Thus any computer program gives rise to a series of instructions adapted to the processor, microprocessor or microcontroller for which it is intended.

Conventionally, the instructions of a program are executed by a processor in a sequence governed by an instruction counter, as will be described briefly with reference to Figure 1.

The compiled instructions of a program are loaded in blocks of successive instructions Inst.1, Inst.2, Inst.3, ..., Inst.n (where n is an integer) in the form of codes or microcodes in an instruction register 2. Each instruction is identified by a specific address in this register 2. In the example, the addresses of the instructions Inst.1, Inst.2, Inst.3, ..., Inst.n are designated respectively Ad.1, Ad.2, Ad.3, ..., Ad.n. The instructions are read from the instruction register 2 and loaded successively into the processor 4 in order to be executed therein under the control of an instruction counter 6, itself controlled by the processor 4. For this purpose, the instruction counter 6 has an address pointer 8 which designates the address Ad.1, ... Ad.n of the register 2 from which the instruction to be loaded into the processor 4 during the execution of an instruction sequence must be read. The position of the pointer 8 vis-à-vis addresses in the instruction register 2 therefore changes along with the execution of the instructions.

In the example depicted in Figure 1, the instructions Inst.1, Inst.2, Inst.3, ..., Inst.n in the register 2 must be executed successively from the first instruction Inst.1 to the nth instruction Inst.n, that is to say in a linear fashion. Thus the pointer 8 of the instruction counter 6 designates initially the address Ad.1 of the register 2, and the data 10-1 of the corresponding instruction Inst.1 are loaded into the processor 4. When the processor 4 orders the instruction counter 6 to supply the next instruction (in this case Inst.2), the said counter increments the position of the pointer 8 by one address change unit in order to designate the address Ad.2. This process is repeated and terminates when the pointer 8 designates the address Ad.n in order to load the data 10-n of the last instruction Inst.n (dotted lines).

A series of instructions executed in a linear fashion does not contain any "jumps" which escape from the sequential progression of the pointer 8 vis-à-vis successive addresses. This is the case, for example, with a sequence of instructions in microcodes such as:

```
lda     t
txa
mul
bset    3,t
sta     n
```

In other words, this sequence will be executed in a linear fashion, the instruction counter 6 being incremented by one address change unit when passing from one instruction to another.

However, it is normal for the execution of a program to require jumps to instructions outside the linear sequence of the instructions as presented in the register 2. Such jumps can result from an instruction for loading data situated in an address outside the sequence, or from an instruction, referred to as a switching instruction, for conditional execution of the following command.

An instruction which causes a jump makes it necessary, at the instruction counter 6, to determine the address in the register 2 of the next instruction following this jump and to position the pointer 8 at this address so that the instruction or the data situated there is loaded into the processor 2.

By way of example, the sequence:

```
lda
txa
bra     label           ;jump
mul
bset    3,t
sta     n
label   rts
```

will cause the loading of a new value into the instruction counter 6 at the position in the code corresponding to the comment "jump".

The fact of thus being able to make, under command, jumps to addresses outside a sequential series of addresses may unfortunately give rise to unacceptable steps in the program. Such unacceptable steps in the program may result from an accidental malfunctioning of the program device. However, it may also result from a malevolent action aimed at diverting the functioning of the program device from its required function. For example, in the case of a smart card, modifications to the programming of the microprocessor by creating or modifying jumps and/or switches may make it possible to load erroneous data (increasing credit authorised with a bank or telephone card, false authorisation of access to certain services, etc) or to recover confidential data stored in memory (access code, personal information concerning the holder of the card, etc).

This is because, even when the programs are recorded in a fixed manner on a semiconductor chip, it is possible, with current techniques for probing and testing components, to create probing contacts on the surface of the chip (or even in the low layers thereof) using work stations with focus ion beams or FIBs.

Once created, these probing points make it possible to deposit probing spikes using specialised workbenches, also known by the English term "probe stations", allowing the continuous reading of a bit (and the monitoring of its change over time) or an external modification of its value.

In particular, depositing contacts on the register of the instruction counter 6 makes it possible to externally load the value of the register currently being executed and to cause a switching not provided for by the program designer. As explained above, such a jump may naturally have harmful consequences for the security of the application and, for example, result in the disclosure of secret data whilst effecting incomplete calculations.

There also exist more rudimentary, but less certain, ways of causing such malfunctioning in the running of the program. One example is given in an article entitled "Tamper resistance, a cautionary note" by R. Anderson. Another technique consists of exploiting calculation errors deliberately caused in order to extract, from a smart card, data such as the secret keys. This technique is described in the article entitled "On the Importance of Checking Computations" by Boneh, DeMillo and Lipton; Bellcore Report, published on 31 October 1996.

Naturally, the same phenomenon can also occur when the program attacked is interpreted instead of being compiled. Thus an application in Java or Basic language can be diverted from its legitimate use if the attacker manages, for example, to cause a change in the program pointer of the interpreter indicating the current instruction to the interpreter.

However, current computer systems are not specifically designed to prevent uncontrolled switchings within a code. Quite the contrary, the assembly languages have been specifically designed to allow the maximum freedom to the programmer. By way of example, in C language, it is possible to jump within the code of a function using the execution indexed by the pointer.

In the light of these problems of running a program in an unacceptable manner, whether they are caused by unwanted malfunctioning or an intention to divert the program from its expected use, the invention proposes a method for monitoring the progress in execution of a series of instructions of a computer program, consisting of analysing the sequence of instructions transmitted to the processor intended to execute the program being monitored and to verify the result of this analysis with reference data recorded with the said program.

Thus the present invention makes it possible to verify that all the instructions included in the set of instructions under consideration have indeed been transmitted to the processor with a view to their execution. It is to be assumed that, if such is the case, the instructions read thus will also have been executed.

The reference data can for example be a value pre-established so as to correspond to the result of the analysis effected during the monitoring method only if all the instructions in the sequence of instructions have actually been analysed during the running of the program.

Preferably, the analysis step comprises the substeps of extracting a data item from each instruction transmitted to the processor and of predetermined calculation on each data item thus extracted, and the verification step includes the comparison of the result of the analysis with the reference data.

Advantageously, the verification step is effected by a hard-wired comparison of a value contained in a register associated with the monitoring means with the reference value, the latter being able to be entered in the program in a hard-wired manner, fixed (for example in a fixed memory of the ROM type) once and for all during the masking of the code which constitutes the program being monitored.

Preferably, the verification is caused by an instruction placed at a predetermined location in the program, this instruction containing the aforementioned reference data.

Advantageously, when the instructions in the set of instructions to be monitored are in the form of a value, hexadecimal or decimal, these instructions are treated as simple numerical values during the aforementioned analysis.

The global method for monitoring the execution of a sequence of instructions of a computer program can thus comprise the following steps:

- during the preparation of the program:
    - incorporating, at at least one predetermined location in a sequence of instructions in the program, a reference value established according to a given rule applied to identifiable data in each instruction to be monitored, and
- during the execution of the part of the program to be monitored:
    - obtaining the said identifiable data in each instruction executed,

- applying the said given rule to the said identifiable data thus obtained in order to establish a verification value, and
- verifying that this verification value actually corresponds to the reference value recorded with the program.

In a preferred embodiment of the invention, provision is made for interrupting the running of the program when it is detected that the verification value does not correspond to the reference value. This interruption can be accompanied by an invalidation action for future use of the device comprising the computer program monitored if the non-correspondence between the verification value and the reference value is detected a predetermined number of times.

Advantageously, the set of instructions to be monitored does not include jumps in its running provided for, so that it is expected for all the instructions which it includes to be executed in all cases envisaged.

When the program or portion of program to be monitored provides for at least one jump, it is possible to apply the monitoring method separately to the sets of instructions which do not include jumps.

In the case of an instruction giving rise to at least one jump dependent on the data being manipulated, that is to say a conditional switching, it is possible to use the monitoring method separately for a set of instructions with no jumps and which precedes the jump, and for at least one set of instructions with no jumps which follow this jump.

In this case, it can be envisaged that, for a set of instructions preceding a jump, there is integrated in this set the instruction which demands the jump (this instruction normally being the last one before a switch) for the purpose of the analysis aimed at obtaining the verification value of this set of instructions, and the correct running of this set of instructions is thus verified before executing the jump instruction.

Advantageously, the verification value obtained during a previous implementation of the method is erased at each new implementation of the method. This provision makes it possible to easily manage the monitoring of different sets of instructions in a program, such as those separated by jumps. It makes it possible notably to implement the method with the same initial conditions of verification value calculation for the different sets separated by jumps.

At each new implementation of the method, the verification value can be erased by a simple resetting. This value can also be replaced by another predetermined initial value. These resetting or initialisation operations can be activated by the protected software itself.

Advantageously, the verification value is obtained as the last value of a series of values which are made to change successively with the analysis of each of the relevant instructions of the set of instructions. This approach makes it possible to contain an internal state of the running of the monitoring method and to follow its changes.

Preferably, the analysis mode allowing this change in the verification value consists of calculating, for each relevant instruction following on from a previous instruction, the result of an operation on both a value extracted from the instruction in question and the result obtained by the same operation performed on this previous instruction. For the calculation related to a first instruction to be verified, it is possible to apply an operation to both the data extracted from this first instruction and a predetermined value (which may then correspond to the reinitialisation value or the aforementioned resetting value), this being able to serve as a "seed" value in the absence of a result of a previous operation.

In this way, it is possible to obtain the correct verification value using a recursive algorithm applicable in the same way for the data extracted from each of the instructions in question. What is more, the calculation operation can easily be chosen so that a correct verification value is obtained only if some of the data of all the instructions have been considered during the calculation, and moreover they have been considered in the order provided for.

The calculation operation can consist in applying a hash function, according to a technique known per se in the field of data enciphering, such as the SHA-1 hash function established by federal hash standard. In this case it is possible to effect the aforementioned internal change in the running of the monitoring method by cryptographically hashing all the operating codes (considered as numerical values) and the addresses executed since the last initialisation carried out.

In a variant, it is possible to change the verification value by making a redundancy calculation, not necessarily cryptographic, on all the operating codes and addresses executed since the last initialisation made. By way of example, it is possible to use algorithms of the CRC (cyclic redundancy check in English) type.

It is possible with the invention to obtain the comparison value by the calculation of intermediate values as the data included in the respective instructions are obtained during the execution thereof. With this approach, it is not necessary to save each value extracted from the instructions of the set of instructions in question. This is because, at the end of a calculation of intermediate value, only this intermediate value counts for calculating the next intermediate value (or the last value, which corresponds to the verification value), and the data item which made it possible to generate it is no longer to be taken into consideration. This arrangement makes it possible to save on memory space with regard to the means of implementing the invention.

As a variant, it is possible to save each data item included in the instructions of the set of instructions considered as they are executed and to effect the calculation of the verification value only at the necessary time, for example at the verification step.

The invention also relates to a device for monitoring the steps of execution of a series of instructions of a computer program, characterised in that it has means for analysing the sequence of instructions transmitted to the processor intended to execute the program monitored and means for verifying the result of this analysis with reference data recorded with the said program.

The monitoring device according to the present invention advantageously includes a register for recording intermediate results in the calculation of the verification value. This register can be adapted to retain only the last current intermediate result.

Provision can be made for allowing the recording of a predetermined value or a resetting under the command of the program currently being executed. In this way, the program can demand an initial condition with regard to the content of the register at each new implementation of the monitoring method, this acting for example after a jump in the program.

The monitoring device can be integrated into a device for executing a program to be monitored or into a programmed device which contains the program to be monitored.

The invention also relates to a program execution device, for example a computer, an appliance with a microprocessor or microcontroller such as a smart card reader or a reader for a program recorded on a card to the PCMCIA format, intended to execute a series of instructions of a computer program, characterised in that it has means for analysing the sequence of instructions transmitted for execution and means for verifying that the result of this analysis corresponds with reference data recorded with the program.

The invention also relates to a program device intended to function with the aforementioned program execution device and including a series of instructions, characterised in that it also includes reference data pre-established according to data contained in the said instructions and intended to allow a verification of the sequence of instructions analysed by the aforementioned program execution device.

The programmed device, for example a smart card or a mechanism control device, such as an ABS braking system, can include the program to be monitored in a fixed memory of the ROM type.

The reference data are advantageously recorded in the form of prewired values fixed in the memory once and for all during the masking of the code.

The present invention also relates to a device for programming a device for executing a program intended to function in association with the aforementioned programmed device, characterised in that it comprises means for entering, at at least one predetermined location in a series of instructions of the program, a reference value calculated according to a pre-established mode from data included in each instruction in a set of instructions whose execution it is wished to monitor.

Finally, the invention also relates to a virtual machine or interpreter interpreting a critical code, characterised in that it implements the aforementioned monitoring method for executing this critical code.

The aforementioned devices for monitoring, executing a program or programming or the devices equipped with such programs can be equipped with all the means necessary for achieving the different possible optional aspects of the aforementioned monitoring method.

By way of example, it is possible to envisage, in an application related to a smart card, to add to a microprocessor executing a program, an additional hardware component serving as a monitoring unit. The role of this unit is to monitor that any jumps not provided for by the software designer cannot take place during execution. In this example, the monitoring unit can be composed of registers whose content constitutes at any time the internal state of the monitoring unit. A specific input of the monitoring unit enables it to be reset, typically by erasing the content of the monitoring unit. This function can be activated at any time by the software being executed and can, for example, be effected by the addition of a new operating code in assembler (for example "clr us") or by manipulating a data bit in the memory of the protected component (for example: setb 3, service).

In this example application, the monitoring unit compares its internal state with a data string supplied by the protective software. This can for example be effected by copying, inside the monitoring unit (by means of a loop "lda-sta") the value at which the required software compares the internal state. Once the copying of the value has ended, the monitoring unit compares it with its internal state and adopts the following behaviour: if the state of the monitoring unit is equal to the value presented by the protected software, resuming the execution normally, otherwise the execution of the program is stopped (forcing the user to reset the card), possibly by previously ratifying a false execution counter in a non-volatile memory of the EEPROM type having the effect of definitive blocking of the card if its value exceeds a reasonable limit (for example 4).

The monitoring unit can permanently keep a cryptographic hash of the instruction code and addresses executed since its last resetting.

The monitoring mechanism can be adapted to the interpretation of code in a virtual machine (of the Java "byte code" type for example). The compiler can calculate the value of the hash of a portion of byte code, integrated into an attribute of a structure known by the English term product "class file" and add to the byte code generated codes known by the English term additional "opcodes" corresponding to the resetting of the monitoring unit and to the invocation of the verification function. The virtual machine will take the place of a monitoring unit and, when it encounters the verification opcode, will verify the value of the current hash with respect to the value of the theoretical hash contained in the class file.

The invention will be more clearly understood and the advantages and characteristics will emerge more clearly from a reading of the following description of a preferred embodiment, given purely by way of example, with reference to the accompanying drawings, in which:

- Figure 1, already presented, is a simplified block diagram aimed at explaining the role of an instruction counter in the execution of a program,

- Figure 2 is a simplified block diagram of a program execution device aimed at explaining the operating principle of a monitoring unit in accordance with a first embodiment of the invention,

- Figure 3 is a flow diagram of the monitoring method according to the invention,

- Figure 4 is a flow diagram of a variant of the monitoring method according to the invention,

- Figure 5 is a simplified block diagram of a program execution device aimed at explaining the operating principle of a monitoring unit according to a second embodiment of the invention,

- Figure 6 is a flow diagram of the monitoring method according to the invention, adapted to the second embodiment, and

- Figure 7 schematically depicts sets of instructions of a program with switching also containing instructions specific to the monitoring method.

The principle of the invention will be explained with reference to the block diagram of Figure 2, in which the blocks having a role similar to those of Figure 1 bear the same references and will not be described again for reasons of conciseness.

Figure 2 depicts the basic elements of a program execution device 20 in the broad sense of the word. It can be a computer intended to execute a program in a high-level language, a microprocessor or a microcontroller, the latter functioning from programs in low-level language. By way of example, the execution device 20 can be a smart card reader intended to manage banking or telephone transactions or other services. The program to be verified is then physically contained in the smart card.

In order to make the following description concrete, it will be assumed that the program execution device 20 is based on a processor 4 of the microcontroller type.

The processor 4 executes a portion of program stored in an instruction register 2 in the form of microcodes. The operational part of this program portion comprises a sequence of n instructions (where n is an integer greater than 1) designated respectively Inst.1, Inst.2, Inst.3, ..., Inst.n. The microcodes which constitute the instructions are in the form of numerical values, which can be decimal or hexadecimal.

Such a value can thus be considered in two different ways, each carrying a data item: firstly as an instruction which it represents for the processor (in which case it will be designated "code value"), and secondly as a simple numerical value capable of arithmetic processing (in which case it will be designated "numerical value" Vinst.). For example, the first instruction Inst.1 is equal to 40. This figure is a code which corresponds to an instruction recognised by the processor, but it has the same binary structure as the numerical value 40.

None of the instructions Inst.1, Inst.2, Inst.3, ..., Inst.n orders a jump to another instruction outside the linear sequence of execution of the instructions. Thus the normal and expected running of this program portion necessarily requires the execution of each of the instructions in succession, commencing with the instruction Inst.1 and ending with the instruction Inst.n. In this way, the instruction counter 6 (already described) will position its pointer 8 successively at the address of each of the instructions Inst.1 to Inst.n in the instruction register 2 as they have to be loaded into the processor 4.

In accordance with the present invention, the program execution device 20 has a monitoring unit 22 which makes it possible to verify that each of the instructions Inst.1 to Inst.n has indeed been loaded to the processor 4 with a view to their execution. It is functionally connected between the instruction register

2 and the processor 4. Thus all the instructions read from the instruction register 2 pass through the monitoring unit 2 before arriving at the processor 4.

In this example, the monitoring unit 22 is described as being integrated into the program execution device 20. However, the monitoring unit 22 can just as well be integrated with the device which includes the program to be monitored, whilst for example being incorporated in a smart card where the program which it contains in memory is to be monitored, without this changing the principles which will be described below.

As will be explained in more detail below, the monitoring unit 22 has a register 24 intended to temporarily store a data item included in the instruction Inst.1, Inst.2, Inst.3, ..., Inst.n and a calculator 26 intended to execute an operation on this data item.

Use of the monitoring unit requires the addition of two new instructions to the n instructions Inst.1, Inst.2, Inst.3, ..., Inst.n of the program. A first monitoring instruction Inst.0 placed before the first instruction Inst.1 of the program and a second monitoring instruction Inst.n+1 placed following the last instruction Inst.n of the program.

During the execution of the n instructions of the program, the instruction counter 6 is initially controlled so as to position its pointer 8 at the address of the first monitoring instruction Inst.0. This instruction instructs the monitoring unit to initialise a hash value VH contained in its register 24. In the example, the instruction Inst.0 simply instructs the register 2 to put the value VH = 0. It is not transmitted to the processor 4.

Next, the program execution device 20 goes into the phase of executing the instructions Inst.1, Inst.2, Inst.3, ..., Inst.n of the program proper. Each instruction read from the instruction register 2 is transmitted firstly to the monitoring unit 22, where it is considered as a numerical value.

The numerical value of each instruction is subjected by the calculator 26 to a hashing algorithm, such as SHA-1 hashing specified by the federal hashing standard. The result VHi of the hashing operation related to an instruction Inst.i (where i is an integer from 1 to n) is entered in the register 24.

This value VHi serves as a basis for the hashing operation with the following instruction Inst.i+1. The result of VHi+1 hashing thus obtained for the instruction Inst.i+1 is then entered instead of the hashing result VHi obtained previously.

This procedure is continued for each of the instructions Inst.1, Inst.2, Inst.3, ..., Inst.n which pass through the monitoring unit 22.

When the last instruction Inst.n is executed, the second monitoring instruction Inst.n+1 is loaded into the monitoring unit 22. This instruction has two components: a reference value Vref and a command, intended for the calculator 26, for comparing this reference value Vref with the value of the last hashing result entered in the register 24. This last value therefore corresponds to the hashing result VHn obtained from the numerical value of the instruction Inst.n (equal to 36 in the figure) and the hashing result VHn-1 obtained for the previous instruction Inst.n-1.

Thus, in response to the second monitoring instruction Inst.n+1, the calculator 26 compares the value VHn in the register 22 with the reference value Vref specified in this monitoring instruction.

The reference value Vref is determined during the preparation of the program recorded in order to correspond to the expected value VHn for the result of the successive hashings of the values of the instructions Inst.1, Inst.2, Inst.3, ..., Inst.n. This value Vref can be calculated in advance using the same procedure of successive hashing of the instructions Inst.1, Inst.2, Inst.3, ..., Inst.n as used by the monitoring unit 22.

Preferably, the value Vref is wired in a fixed memory in order not to be able to be modified by a malevolent act.

If the monitoring unit 22 finds, whilst executing the monitoring instruction Inst.n+1, that there is identity between the aforementioned values Vref and VHn, it is concluded that all the instructions Inst.1, Inst.2, Inst.3, ..., Inst.n have indeed been transmitted to the processor 4 with a view to their execution.

If, on the other hand, the monitoring unit 22 finds that there is not identity between the values Vref and VHn, it is concluded that either not all the instructions Inst.1, Inst.2, Inst.3, ..., Inst.n have been received and transmitted by the monitoring unit, or they have not been received and transmitted in the expected sequential order. In this case, an action can be provided for aimed at alerting the user or owner of the program, or preventing the program from continuing. In the example, such an action is transmitted from the monitoring unit 22 to the processor 4 in the form of a command for interruption of the program Int.

The monitoring method as implemented by the means of Figure 2 will now be described with reference to the flow diagram depicted in Figure 3. It is assumed that the program or the program part to be monitored was prepared correctly for the monitoring method by the incorporation of the first and second monitoring instructions respectively at the start and end.

At the initial stage, the monitoring unit 22 is positioned on a start of monitoring routine 30, the first step 32 of which is the awaiting of the first monitoring instruction (Inst.0).

When the first monitoring instruction Inst.0 is received, the monitoring unit 22 effects a step 34 of initialisation (by resetting) of an instruction counter and the register 24. The resetting of the register 24 is a way of placing a "seed" value in this register in order to start a sequence of hashing operations, as will be explained later. These operations can be controlled directly by the first monitoring instruction or simply be triggered by it from a routine associated with the monitoring unit 22.

In this first case, the resetting can be effected by the addition of a new operating code in assembler (for example "clr us"), or by manipulating a given bit in the memory of the program execution device 20. Such a command can be "setb 3, service".

After the initialisation step, the monitoring unit 22 increments the instruction counter by one unit (then positioning this counter to n=1) (step 36).

Next, the first instruction Inst.1 of the program or the program part under surveillance is read from the instruction register 2 (step 38). As explained above, this instruction is considered by the monitoring unit 22 purely as a numerical value allowing arithmetic operations. In the example in Figure 2, this value is 40.

The digital value of this first instruction Inst.1 is then subjected to a hashing operation with the value contained in the register 24 (step 40). In the case of the first instruction, this last value is the initialisation value, that is to say 0.

The hashing operation, well known per se, consists here of making a mathematical operator f(VHn-1, Vinst.n) act on the value of the instruction n in question, where VHn-1 is the result of a previous hashing operation (or the initialisation value in the case of the first instruction) recorded in the register 24 and Vinst.n is the numerical value of the instruction n in question.

The result VHn of this hashing operation is then recorded in the register 24 in place of the previous result VHn-1 (step 42). It should be noted that this procedure of updating the content of the register at each hashing operation makes it possible to permanently keep a cryptographic hash of the instruction codes and addresses executed since the last initialisation.

At the end of this hashing operation, the instruction is transmitted to the processor 4 with a view to its execution (step 44).

Next the monitoring unit 22 determines whether the program or the program part to be monitored contains another instruction to be executed (step 46).

This being the case, the procedure effects a looping back B1 to the step 36 of incrementation from n to n+1. The value of the next instruction (Inst.2) will then be read from the instruction register2 and subjected to the hashing operation in the same way as for the instruction Inst.1. However, the hashing takes place this time with on the one hand the numerical value of the instruction Inst.2 and the result obtained during the previous hashing operation, that is to say the value VH1 (n being here equal to 2), which is then in the register 24.

Steps 42 to 46 of the method take place in the same way as for the first instruction. In this way, the set of steps 36 to 46 continues in a loop for each instruction Inst.1, Inst.2, Inst.3, ..., Inst.n read from the instruction register 2, with the hashing taking place, for an instruction Inst.i (where i is an integer from 1 to n) with the value VHni-1 in the register 24 and the value Vinst.i.

Once all the instructions Inst.1, Inst.2, Inst.3, ..., Inst.n have been thus processed by the monitoring unit 22, the latter receives the second monitoring instruction Inst.n+1, this following the last instruction Inst.n of the program or of the program part being monitored.

This second monitoring instruction instructs the monitoring unit 22 to extract the reference value Vref from the program (step 48) and to compare the content of the register 24 with this value Vref (step 50). This command can be effected by means of a loop "lda-sta".

It should be stated that, by the execution of successive loops of steps 36 to 46, the value contained in the register 24 at this stage is the result VHn of the hashing carried out with the result VHn-1 of the previous hashing and the numerical value of the instruction n (equal to 36 in the example in Figure 2).

The reference value Vref was previously determined during the preparation of the program with knowledge of the hashing operations, in order to be equal to what the monitoring unit 22 should return as the value VHn if all the instructions Inst.1, Inst.2, Inst.3, ..., Inst.n were indeed transferred to the processor 4.

Thus the result of the comparison makes it possible to check that the instructions Inst.1, Inst.2, Inst.3, ..., Inst.n have unfolded correctly: if VHn =

Vref (step 52), it is assumed that all the instructions have actually been transferred to the processor 4. The monitoring operation is then terminated with regard to the program or program portion containing the instructions Inst.1, Inst.2, Inst.3, ..., Inst.n.

The monitoring procedure then returns to the starting phase 30 awaiting a new first monitoring instruction.

If on the other hand the comparison step 50 reveals that there is no identity between the compared values (VHn≠Vref), it is assumed that not all the instructions Inst.1, Inst.2, Inst.3, ..., Inst.n have been transferred to the processor 4, or have not been transferred in the correct order (step 54). This is because the result of a succession of hashing operations depends on the order in which they were carried out.

In this case, the monitoring unit 22 demands an action (step 56), such as the interruption of the program and/or of the recording of the fact that the program has not run correctly.

A variant of the aforementioned monitoring method will now be described with reference to the flow diagram in Figure 4. According to this variant, instead of performing a hashing operation during the reception of each new instruction by the monitoring unit 22, all the hashing operations are performed only after reception of the second monitoring instruction. In the flow diagram in Figure 4, the steps which are identical to those previously described with reference to Figure 3 bear the same references and will not be described again for reasons of conciseness.

The monitoring method unfolds as for the previous case with regard to steps 30 to 38 (Figures 3 and 4). After the step 38 of reading the value of the instruction Vinst.n of the instruction register 2, the monitoring unit 22 proceeds with the recording of this value (step 39). This recording can take place in an internal register of the calculator 26, in a dedicated section of the register 24, in a specific memory (not shown) of the monitoring unit 22, or in a memory external to the monitoring unit 22, provided that it is accessible to the latter.

Next the monitoring unit 22 proceeds with the previously described steps 44 and 46. It will be noted that the step 39 of recording the values Vinst.n is in the loop B1 connecting step 46 to the step 36 of incrementing n by one unit, so that each of the values Vinst.n is thus recorded up to the detection of the second monitoring instruction at step 46.

When this second instruction appears, the monitoring unit 22 reads the reference value Vref (step 48) and performs at step 49 the hashing according to the same algorithm as at the previously described steps 40 and 42 of Figure 3, based on the set of values Vinst previously recorded. The final hashing value VHn is then the same as in the case of the method in Figure 3. It should be noted that it is possible to reverse the order of steps 48 and 49.

The comparison steps 50 and those which follow are identical to those in Figure 3.

Figure 5 is a simplified block diagram of the monitoring unit 22 in accordance with a second embodiment of the invention. Its integration into the program execution device 20 is the same as with the first embodiment described with reference to Figures 2 and 3 and with regard to its functioning with the instruction counter 6, the instruction register 2 and the processor 4, and will not be repeated for reasons of conciseness.

The monitoring unit 22 according to the second embodiment is distinguished from that of the first embodiment essentially by the fact that it also has a memory 60 which records the number of times the execution of a series of instructions Inst.1-Inst.n has not taken place correctly in accordance with the criteria explained with reference to Figure 3 or 4.

In the example, the memory 60 is produced in the form of a fixed (non-volatile) memory with electrically erasable content (commonly known by the English term EEPROM).

The memory 60 is functionally connected to the calculator 26 so that it records a counting value VC incremented by one unit each time an incorrect execution is noted in the series of instructions being monitored. This counting value VC thus makes it possible to detect the number of incorrect performances of the series of instructions and to act accordingly, for example by invalidating any future use of the device containing the program (for example a smart card) if this number passes a threshold value.

The flow diagram in Figure 6 gives an example of use of the counting value VC for monitoring the device of the execution of the program. This example includes the set of steps 30 to 54 of the flow diagram in Figure 3 or the analogous steps in Figure 4.

When the monitoring unit 22 detects at step 54 an unexpected execution of the instructions Inst.1-Inst.n, following the comparison step 52, the calculation unit 26 increments the counting value VC in the memory 60, initially 0, by one unit (step 62). Next it checks whether the counting value VC thus incremented has reached a predetermined threshold value VCthreshold (step 64). This threshold value VCthreshold corresponds to the number of times it is accepted that the unexpected execution of the instructions Inst.1-Inst.n can occur in the programmed device before taking definitive measures for dealing with such a fault. By way of example, in the context of a smart card, a moderate number of such failures can be accepted (for example 3 or 4) to the benefit of the doubt as to whether it could be a case of a momentary damage related to the reader (program execution device 20), but beyond this number it must be considered that the card has been impaired either accidentally or by malevolence. In this case of implementation, provision can also be made for entering the value VC in the programmed device (card) in order to keep a history of these faulty executions related physically to the program device.

If the counting value VC is below the threshold VCthreshold, the monitoring unit 22 establishes the interrupt command Int as previously described with a simple alert message intended for the user and/or the operating system (step 66) and transmits it to the processor 4 (step 68).

If on the other hand the counting value VC reaches the threshold value VCthreshold, the monitoring unit 22 then establishes the interrupt command Int as previously described with an instruction inhibiting any future use of the programmed device containing the instructions executed in an unexpected manner (step 70) and transmits it to the processor (step 68). In this case, it will be possible to reuse this device only after having reprogrammed the memory 60. Where this memory 60 is in the form of an EEPROM or other non-volatile memory, such a reprogramming is very difficult to perform in a diverted manner.

It will be noted that the program interrupt command Int accompanied by an alert message or future usage invalidation command transmission can be executed either at the processor, or at the monitoring unit 22.

It will now be described, with reference to Figure 7, how the monitoring unit 22 according to the present invention can be used for the monitoring of a program which provides for jumps or switchings.

In the example in Figure 7, the program execution device 20 contains in the instruction register 2 a program or program part intended for the processor 4, consisting of three sets of instructions:

- a first set of instructions Inst.EI1-1 to Inst.EI1-j (where j is an integer > 1), the last instruction EI1-j is a code which demands the conditional switching to one or other of the other two sets which follow;

- a second set of instructions Inst.EI2-1 to Inst.EI2k (where k is an integer > 1); the execution of the first instruction Inst.EI2-1 of this set follows on from the execution of the conditional switching instruction EI1-j if the first of two conditions posed by this is satisfied; and

- a third set of instructions Inst.EI3-1 to Inst.EI3l (where l is an integer > 1); the execution of the first instruction Inst.EI3-1 of this set follows on from the execution of the conditional switching instruction EI1-j if the second of two conditions posed by this is satisfied.

The three sets of instructions EI1, EI2 and EI3 do not include any jumps within their sequence of instructions. (In the case of the first set of instructions, the conditional jump to the instruction EI1-j is at the end of the sequence.) Thus, for each of the three sets of instructions, all the instructions are designed to be executed sequentially as from the first.

During the preparation of the program, there are added at the head and tail of each set of instructions EI1, EI2 and EI3 respectively the first monitoring instruction and the second monitoring instruction described above with reference to Figures 2 to 6.

The monitoring of the program or of the program part composed of sets EI1, EI2 and EI3 then proceeds as follows.

The monitoring unit 22 is first of all positioned in the start of monitoring phase (step 30, Figure 3).

The running begins with the execution of the first set of instructions EI1. The first monitoring instruction placed at the head of this set will first of all be loaded in the monitoring unit 22. In response to this instruction, the monitoring unit initialises its instruction counter and its hashing value VH register 24 (step 34, Figure 3) and proceeds with the hashing routine for each instruction Inst.EI1-1 to Inst.EI1-j of the first set of instructions according to steps 36 to 46 of Figure 3.

The last instruction EI1-j of the set which controls the switching is thus also subjected to hashing by the monitoring unit 22 in order to be transmitted to the processor 4.

The following instruction is the second monitoring instruction (step 46, Figure 3) at the tail of the first set of instructions EI1, which brings about the phase of comparison between the last hashing value recorded in the register 24 with the reference value Vref associated with this second instruction.

If it is detected at the comparison step 50 (Figure 3) that the last hashing value thus recorded does not correspond to the reference value Vref, the monitoring unit 22 proceeds with the program interruption steps 54 and 56 (Figures 3 and 4) or 54 to 70 (Figure 6). Preferably, provision is made for this interruption to occur even before the program has executed the conditional switching. This can be achieved, for example, by associating the jump instruction with an await validation instruction coming from the monitoring unit, employing known programming techniques.

If it is detected at the comparison step 50 (Figure 2) that the last hashing value thus recorded does actually correspond to the reference value Vref, the monitoring unit authorises the execution of the conditional switching determined by the last instruction EI1-j in the set. The program then continues to one or other of the second or third sets of instructions in accordance with the switching conditions posed by this last instruction.

It is assumed in the example that the conditional jump causes the switching to the second set of instructions to be executed. In this case, the instruction counter 6 makes the instruction pointer 8 pass directly from the second monitoring instruction at the tail of the first set of instructions EI1 to the first monitoring instruction at the head of the third set of instructions EI3.

The monitoring unit will execute this new first instruction by reinitialising the instruction counter and the register 24. The monitoring procedure for this third set of instructions therefore continues exactly in the same way as with the first set of instructions. Thus the monitoring unit 22 will proceed with the successive hashing of each of the instructions read in this set, commencing the hashing with the same "seed" value (which corresponds here to zero) as for the first set. The second monitoring instruction this time makes it possible to detect an unexpected course of action in the execution located at this first set of instructions and to proceed with the same type of action at step 56.

It will be understood that the explanation given for the case of switching to the third set applies in a strictly similar manner in the case of a switching to the second set of instructions following the execution of the switching instruction and the first set of instructions.

It is possible to enable the monitoring unit 22 to count not only the number of unexpected steps in a program including jumps, but also the independently monitored sets of instructions in which they are produced.

Thus the monitoring unit 22 according to the second embodiment (Figure 5) can record in its memory 60 the set of instructions to which each interruption noted relates. It is also possible to establish criteria for invalidation of future usage of the program according to the sets of instructions in which the interruptions are situated.

Naturally, it will be understood that the monitoring unit 22 can be either implemented separately from the processor 4 or integrated functionally in it.

Finally, it is clear that all the aspects of the invention described in terms of method can easily be understood in terms of physical means for their implementation, and vice-versa. Likewise, it will be understood that the invention described also covers all the obvious transpositions of one embodiment or its variant to another.

533 Rec'd PCT/PTO 10 SEP 2001
09/936174

5/PRTS

CLAIMS

1. A method for monitoring progress with the execution of a linear series of instructions (Inst.1-Inst.n) in a computer program, consisting in analysing the sequence of instructions transmitted to the processor (4) intended to execute the program being monitored and to verify the result of this analysis by reference to reference data (Vref) recorded with the said program, characterised in that the reference data comprise a value (Vref) pre-established so as to correspond to the result of this analysis produced during the monitoring method only if all the instructions (Inst.1-Inst.n) in the sequence of instructions have actually been analysed during the running of the program and in that the said analysis of the sequence of instructions (Inst.1-Inst.n) comprises the extraction (38) of a data item from each instruction transmitted to the processor (4) and a predetermined calculation (40, 42) on each data item thus extracted, and in that the verification comprises a comparison (50) of the result of the analysis with the reference data (Vref).

2. A method according to Claim 1, characterised in that the said verification of the result of the analysis is caused by an instruction (Inst.n+1) placed at a predetermined location in the program to be monitored, this instruction containing the reference data (Vref) relating to a set of instructions (Inst.1-Inst.n) whose correct execution is to be monitored.

3. A method according to any one of Claims 1 to 2, characterised in that, when the instructions (Inst.1-Inst.n) of the set of instructions to be monitored are in the form of a value, for example codes recorded in hexadecimal or decimal form, the said analysis of the instructions is carried out considering these as a numerical value.

4. A method according to Claim 1, comprising the steps consisting in:

- during the preparation of the program to be monitored:
  - incorporating, at at least one predetermined location in a sequence of instructions (Inst.1-Inst.n) in the program, a reference value (Vref) established according to a predetermined rule applied to identifiable data in each instruction to be monitored, and
- during the execution of the program to be monitored:
  - obtaining (38) the said identifiable data in each instruction received with a view to its execution,
  - applying (40, 42) the said predetermined rule to the said identifiable data thus obtained in order to establish a verification value (VHn), and
  - verifying (50) that this verification value actually corresponds to the reference value recorded with the program.

5. A method according to any one of Claims 1 to 4, characterised in that it also comprises a step (56) of interrupting the flow of the program monitored if the analysis reveals that the program being monitored has not been run as expected.

6. A method according to any one of Claims 1 to 5, characterised in that it also comprises an invalidation step (70) for future use of the device comprising the monitored program if the said analysis reveals a predetermined number of times that the program being monitored has not run in the expected manner.

7. A method according to any one of Claims 1 to 6, characterised in that the set of instructions to be monitored does not include jumps in its expected flow.

8. A method according to any one of Claims 1 to 6, characterised in that, when the program (EI1, EI2, EI3) or the program portion to be monitored provides for at least one jump, the monitoring method is applied separately to sets of instructions in this program which do not include jumps between two successive instructions.

9. A method according to Claim 8, characterised in that, when the program to be monitored includes an instruction (EI1-j) giving rise to a jump dependent on the manipulated data, the monitoring method is implemented separately for a set of instructions (EI1) which precedes the jump, and for at least one set of instructions (EI2, EI3) which follows this jump.

10. A method according to Claim 9, characterised in that, for a set of instructions (EI1) providing for a jump, there is integrated in this set the instruction (EI1-j) which controls this jump for the purpose of the analysis aimed at obtaining the verification value (VH) for this set of instructions, and thus the correct running of this set of instructions is verified before executing the jump instruction.

11. A method according to any one of Claims 1 to 10, characterised in that the analysis is reinitialised before each new monitoring of a sequence or set (EI1, EI2, EI3) of instructions to be monitored.

12. A method according to Claim 11, characterised in that the reinitialisation of the analysis of each new monitoring consists in erasing or replacing a verification value (VH) obtained during a previous analysis.

13. A method according to Claim 11 or 12, characterised in that the reinitialisation of the monitoring analysis is controlled by the protected software itself.

14. A method according to any one of Claims 1 to 13, characterised in that the analysis produces a verification value (VH) obtained as the last value in a series of values which is made to change successively with the analysis of each of the analysed instructions (Inst.1-Inst.n) of the set of instructions, thus making it possible to contain an internal state of the running of the monitoring method and to follow its changes.

15. A method according to any one of Claims 1 to 14, characterised in that the analysis consists in calculating (40, 42), for each instruction under consideration (Inst.n) following on from a previous instruction (Inst.n-1), the result of an operation on both a value (VHn) obtained of the instruction in question and the result (VHn-1) obtained by the same operation performed on the previous instruction.

16. A method according to any one of Claims 1 to 15, characterised in that the analysis consists in recursively applying a hash function f(VHn-1, Vinst.n) to values obtained of each monitored instruction, starting from a last initialisation performed.

17. A method according to any one of Claims 1 to 15, characterised in that the analysis consists in making a verification value change by performing a redundancy calculation, not necessarily cryptographic, on all the operating codes and the addresses executed since the last initialisation carried out.

18. A method according to any one of Claims 1 to 17, characterised in that the analysis consists in obtaining a comparison value (VCn) by calculating successive intermediate values as the data of the respective instructions serving for this calculation during the execution of these instructions are obtained.

19. A method according to any one of Claims 1 to 17, characterised in that the analysis comprises a step of saving each data item necessary for verification, obtained from instructions in the set of instructions

to be monitored (Inst.1-Inst.n) as they are executed, and performing a calculation of the verification value (VHn) from these data only at the necessary time, once all the necessary data have been obtained.

20. A device for monitoring progress with the execution of a series of instructions (Inst.1-Inst.n) of a computer program implementing the monitoring method according to any one of Claims 1 to 19, characterised in that it comprises means (22-26) for analysing the sequence of instructions transmitted to the processor (4) intended to execute the program being monitored and means (26) for verifying the result (VCn) of this analysis by reference to reference data (Vref) recorded with the said program.

21. A device according to Claim 20, adapted to implement the monitoring method according to any one of Claims 1 to 19, characterised in that it has a register (24) for recording intermediate results (VH) in a calculation in a chain carried out by the analysis means (26) in order to obtain a verification value (VHn).

22. A device according to Claim 21, characterised in that it comprises means for allowing the recording of a predetermined value or a resetting of the register (24) under the control of an instruction (Inst.n+1) transmitted during the execution of a program to be monitored, for example on the occasion of a jump in the program.

23. A device according to any one of Claims 20 to 22, characterised in that it has a means (60) for

counting the number of unexpected events in the program being monitored, as determined by the analysis means (26), and means for invalidating the future use of the program to be monitored if this number reaches a predetermined threshold (VCthreshold).

24. A device according to any one of Claims 20 to 23, characterised in that it is integrated into a programmed device, such as a smart card, containing the said program to be monitored.

25. A device according to any one of Claims 20 to 23, characterised in that it is integrated into a program execution device (20).

26. A program execution device (20) intended to execute a series of instructions (Inst.1-Inst.n) of a computer program, characterised in that it has means (22-26) for analysing the sequence of instructions transmitted for execution and means for verifying the result of this analysis by reference to reference data (Vref) recorded with the program to be monitored, according to any one of Claims 20 to 23.

27. A program execution device (20) according to Claim 26, adapted to implement the method according to any one of Claims 1 to 21.

28. A programmed device containing a series of recorded instructions (Inst.1-Inst.n), characterised in that it also contains a fixed memory containing reference data (Vref) pre-established as a function of data contained in the said instructions and intended to allow verification of the sequence of instructions analysed according to any one of Claims 1 to 21, the

said device being intended to cooperate with a monitoring device according to any one of Claims 20 to 27.

29. A device according to Claim 28, characterised in that it is in the form of a smart card.

30. A device according to Claim 30 or 31, characterised in that the reference data (Vref) are recorded in the form of a prewired value or values fixed in memory.

31. A device for programming a device intended to be programmed according to any one of Claims 28 to 30, characterised in that it comprises means for entering, at at least one predetermined location in a sequence of instructions in the program (Inst.1-Inst.n), a reference value (Vref) calculated according to a pre-established mode from data included in each instruction in a set of instructions whose execution it is wished to monitor.

32. A virtual machine or interpreter interpreting a critical code, characterised in that it implements the method according to any one of Claims 1 to 19 for the execution of this critical code.

09/936174

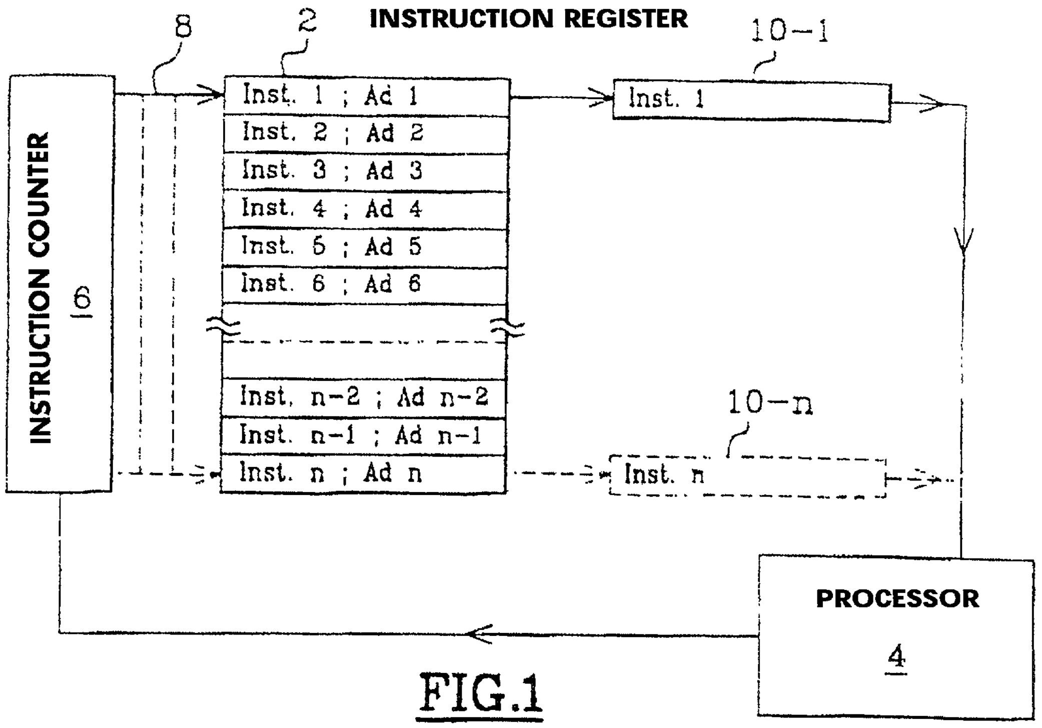


FIG.1

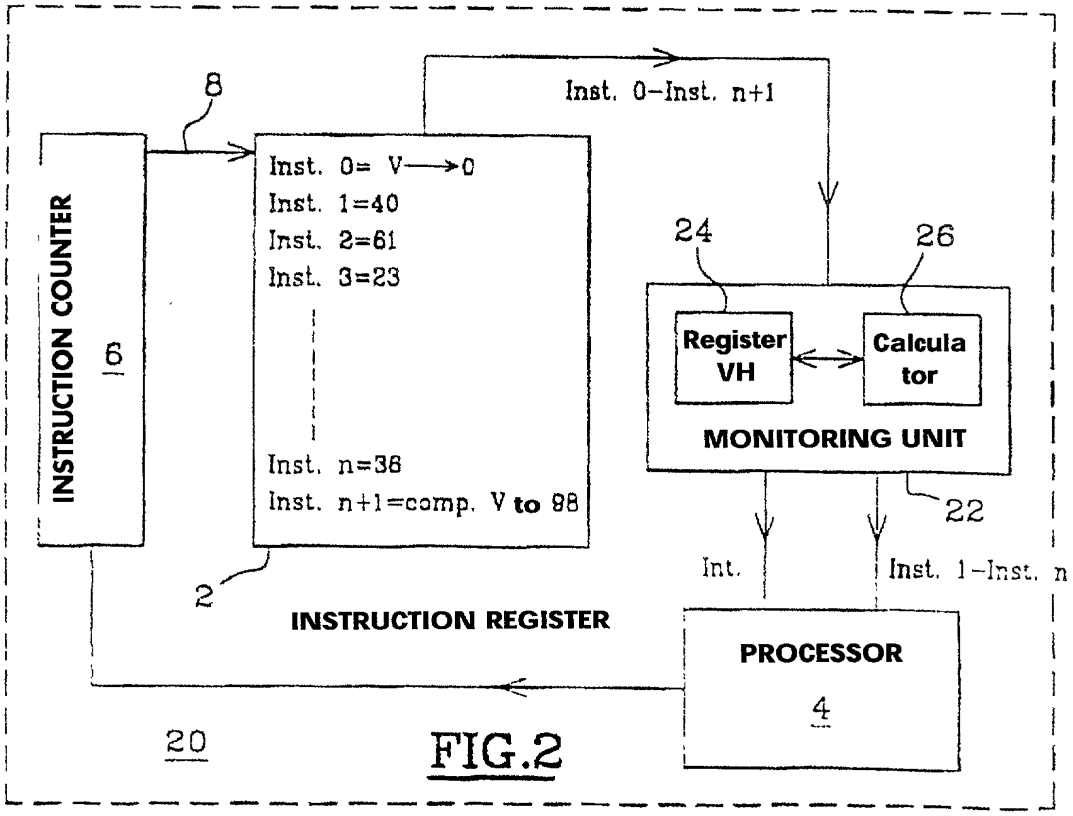


FIG.2

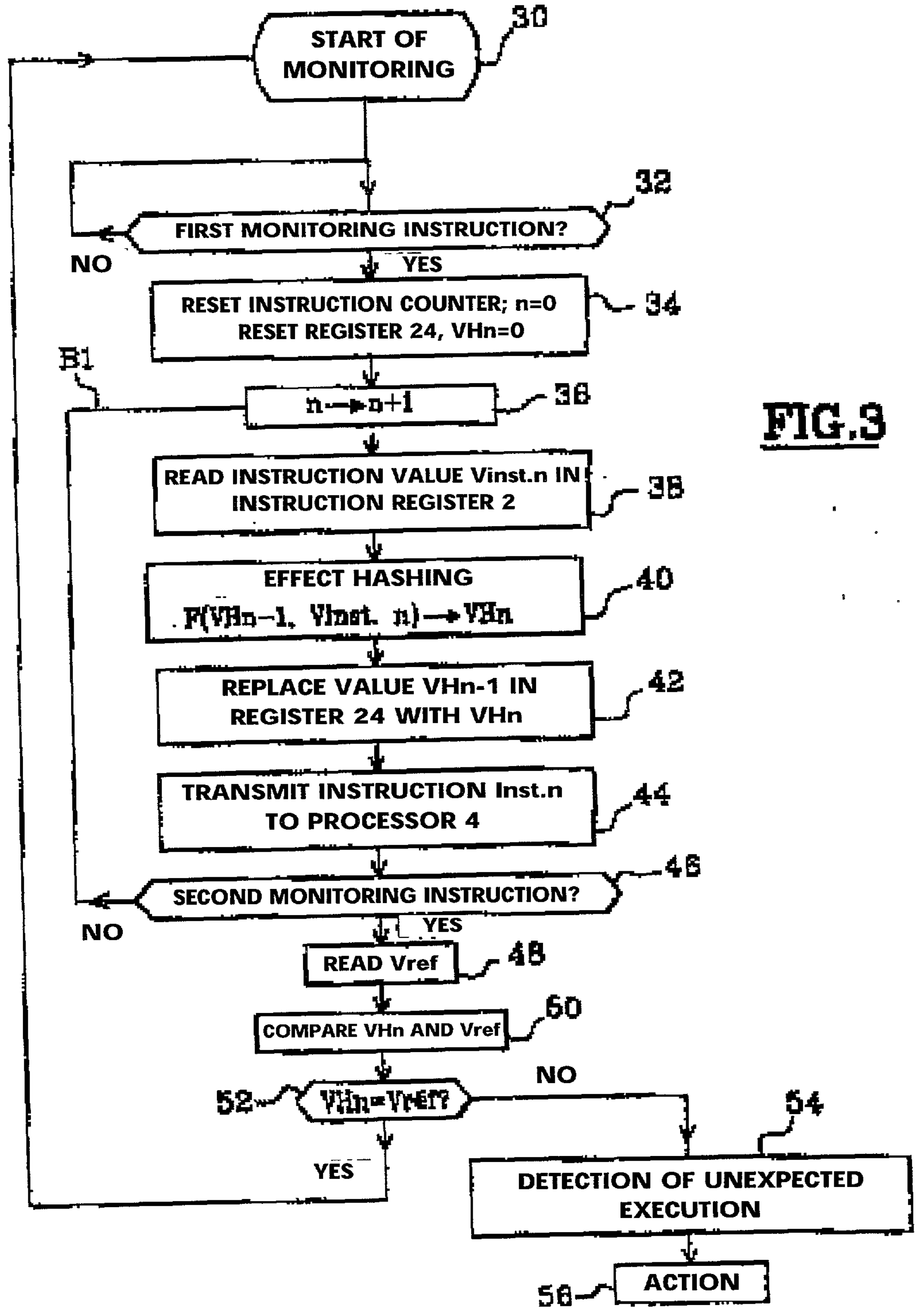
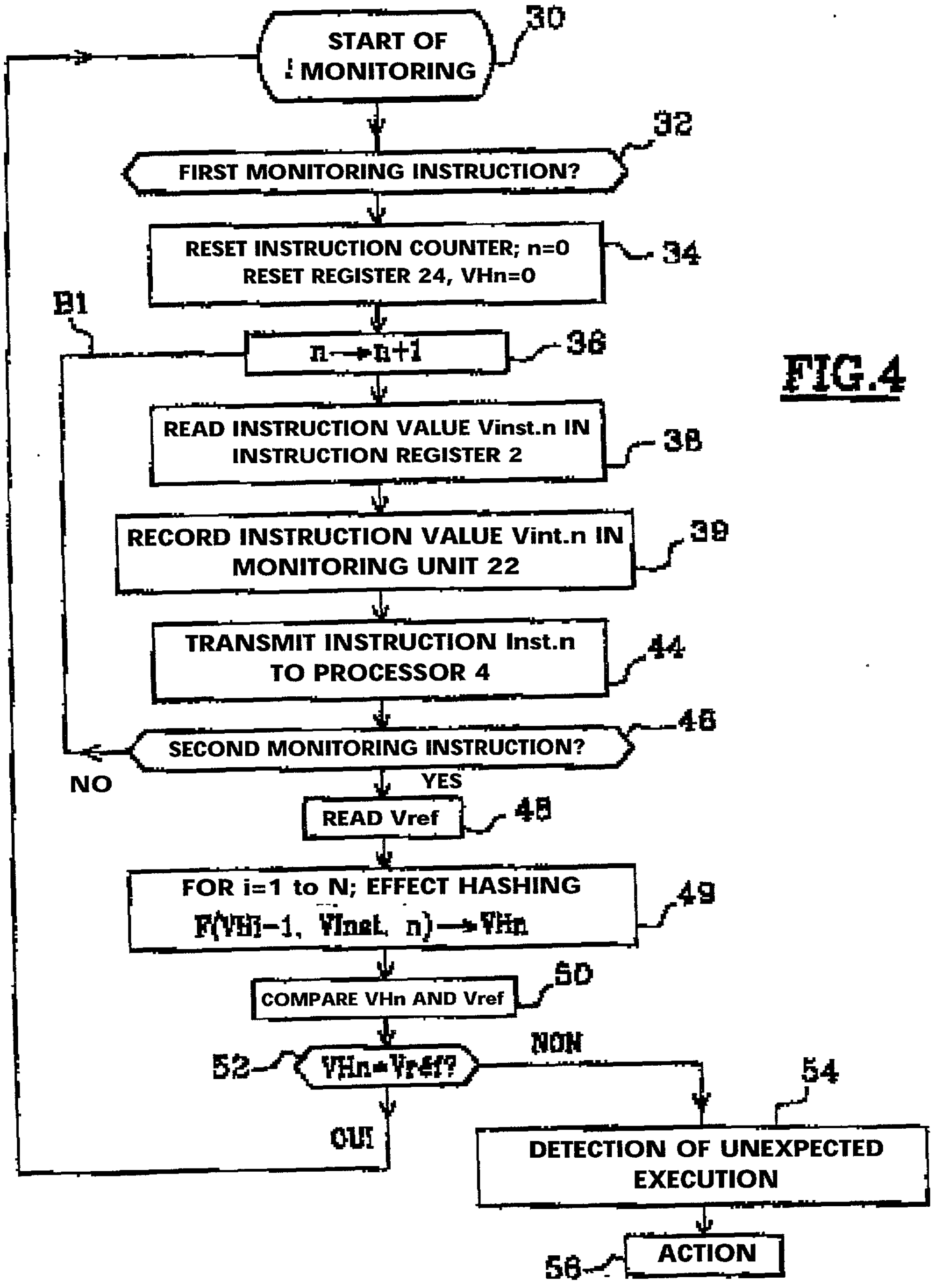

START OF MONITORING
30
FIRST MONITORING INSTRUCTION?
32
NO
YES
RESET INSTRUCTION COUNTER; n=0
RESET REGISTER 24, VHn=0
34
B1
n→n+1
36
READ INSTRUCTION VALUE Vinst.n IN INSTRUCTION REGISTER 2
38
EFFECT HASHING
F(VHn-1, Vinst. n)→VHn
40
REPLACE VALUE VHn-1 IN REGISTER 24 WITH VHn
42
TRANSMIT INSTRUCTION Inst.n TO PROCESSOR 4
44
SECOND MONITORING INSTRUCTION?
46
NO
YES
READ Vref
48
COMPARE VHn AND Vref
50
52
VHn=Vref?
NO
YES
54
DETECTION OF UNEXPECTED EXECUTION
ACTION
56

FIG.3

09/936174



START OF MONITORING
30
FIRST MONITORING INSTRUCTION?
32
RESET INSTRUCTION COUNTER; n=0
RESET REGISTER 24, VHn=0
34
B1
n→n+1
36
READ INSTRUCTION VALUE Vinst.n IN INSTRUCTION REGISTER 2
38
RECORD INSTRUCTION VALUE Vint.n IN MONITORING UNIT 22
39
TRANSMIT INSTRUCTION Inst.n TO PROCESSOR 4
44
SECOND MONITORING INSTRUCTION?
46
NO
YES
READ Vref
48
FOR i=1 to N; EFFECT HASHING
F(VHi-1, Vinst. n)→VHn
49
COMPARE VHn AND Vref
50
52
VHn=Vref?
NON
OUI
54
DETECTION OF UNEXPECTED EXECUTION
ACTION
56

FIG.4

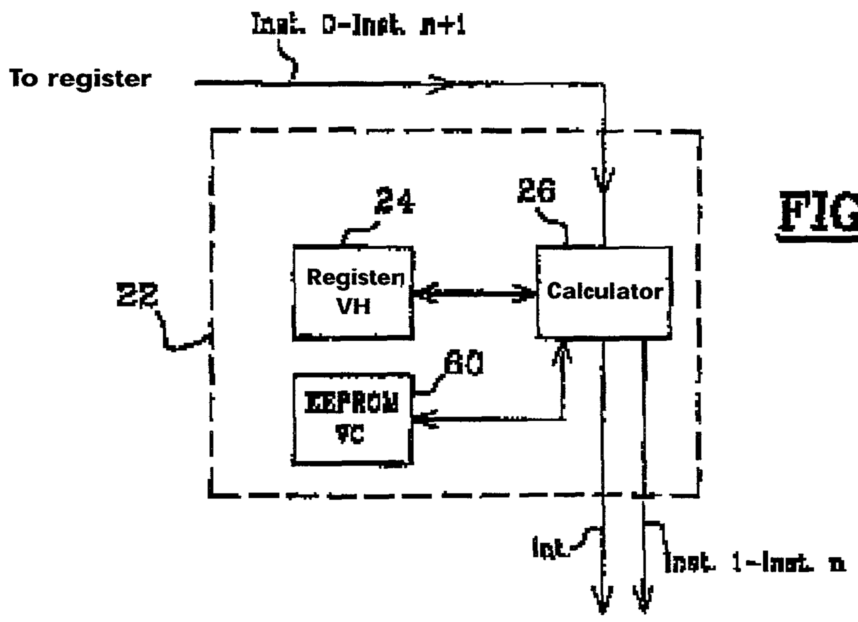

Inst. 0-Inst. n+1
To register
22
24
26
Register VH
Calculator
60
EEPROM VC
Int
Inst. 1-Inst. n

FIG.5

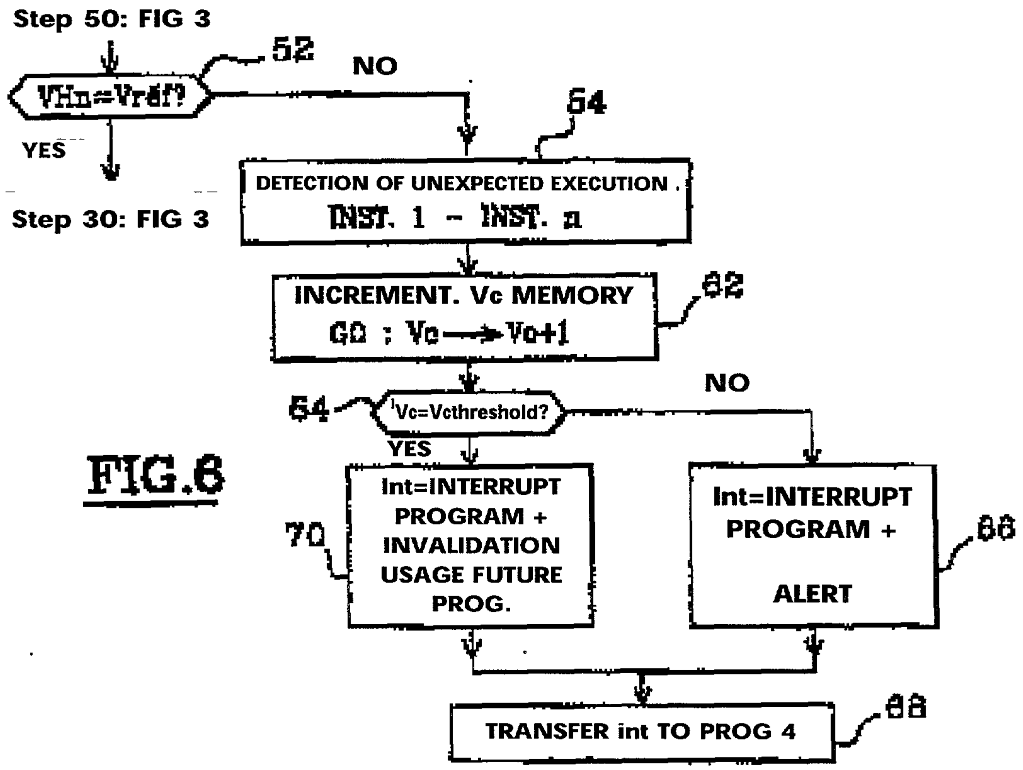

Step 50: FIG 3
52
NO
VHn=Vréf?
YES
Step 30: FIG 3
54
DETECTION OF UNEXPECTED EXECUTION .
INST. 1 - INST. n
INCREMENT. Vc MEMORY
GO : Vc→Vc+1
62
NO
64
Vc=Vcthreshold?
YES
70
Int=INTERRUPT PROGRAM + INVALIDATION USAGE FUTURE PROG.
Int=INTERRUPT PROGRAM + ALERT
66
TRANSFER int TO PROG 4
68

FIG.6

09/936174

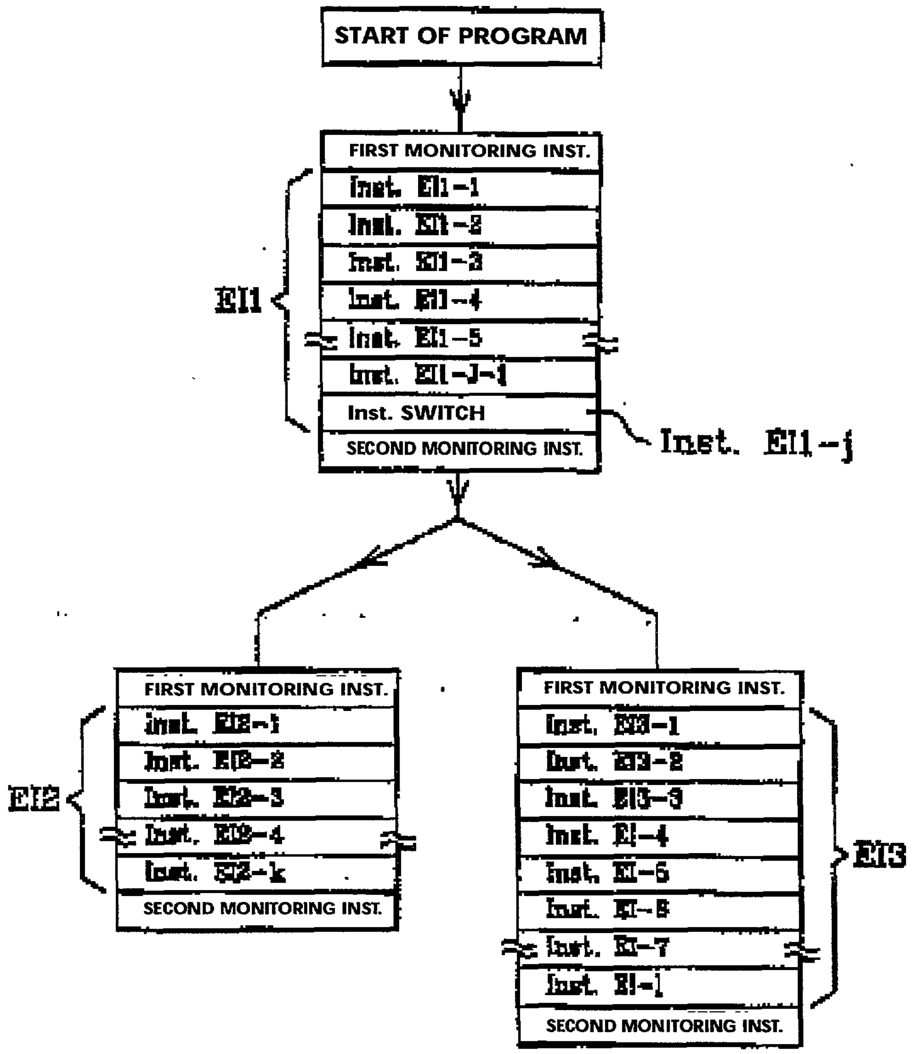


FIG.7

COMBINED DECLARATION FOR PATENT APPLICATION AND POWER OF ATTORNEY (Includes Reference to Provisional and International (PCT) Applications)	Attorney's Docket No. 032326-166

As a below named inventor, I hereby declare that:
My residence, post office address and citizenship are as stated below next to my name;
I BELIEVE I AM THE ORIGINAL, FIRST AND SOLE INVENTOR (IF ONLY ONE NAME IS LISTED BELOW) OR AN ORIGINAL, FIRST AND JOINT INVENTOR (IF PLURAL NAMES ARE LISTED BELOW) OF THE SUBJECT MATTER WHICH IS CLAIMED AND FOR WHICH A PATENT IS SOUGHT ON THE INVENTION ENTITLED:

METHOD FOR MONITORING A PROGRAMME FLOW

The specification of which **(check only one item below)**:

☐ is attached hereto.

☐ was filed as United States Patent Application Number ________
on ________
and was amended on ________ (if applicable).

[✓] was filed as International (PCT) Application Number **PCT/FR00/00150**
on **January 24, 2000**
and was amended on ________ (if applicable).

I HAVE REVIEWED AND UNDERSTAND THE CONTENTS OF THE ABOVE-IDENTIFIED SPECIFICATION, INCLUDING THE CLAIMS, AS AMENDED BY ANY AMENDMENT REFERRED TO ABOVE.

I ACKNOWLEDGE THE DUTY TO DISCLOSE TO THE U.S. PATENT AND TRADEMARK OFFICE ALL INFORMATION KNOWN TO ME TO BE MATERIAL TO PATENTABILITY AS DEFINED IN TITLE 37, CODE OF FEDERAL REGULATIONS, Sec. 1.56 (as amended effective March 16, 1992);

I do not know and do not believe the said invention was ever known or used in the United States of America before my or our invention thereof, or patented or described in any printed publication in any country before my or our invention thereof or more than one year prior to said application; that said invention was not in public use or on sale in the United States of America more than one year prior to said application; that said invention has not been patented or made the subject of an inventor's certificate issued before the date of said application in any country foreign to the United States of America on any application filed by me or my legal representatives or assigns more than six months prior to said application;

I hereby claim foreign priority benefits under Title 35, United States Code, §§ 119 (a)-(e) of any foreign application(s) for patent or inventor's certificate or of any International (PCT) Application(s) designating at least one country other than the United States of America listed below and have also identified below any foreign application(s) for patent or inventor's certificate or any PCT International (PCT) Application(s) designating at least one country other than the United States of America filed by me on the same subject matter having a filing date before that of the application(s) of which priority is claimed:

PRIOR FOREIGN/PCT APPLICATION(S) AND ANY PRIORITY CLAIMS UNDER 35 U.S.C. §119:

COUNTRY (if PCT, indicate "PCT")	APPLICATION NUMBER	DATE OF FILING (day, month, year)	PRIORITY CLAIMED UNDER 35 U S.C. §119
FRANCE	99/02924	March 09, 1999	[✓ Yes] No
			Yes No
			Yes No
			Yes No
			Yes No

I hereby claim the benefit under Title 35, United States Code § 119(e) of any United States provisional application(s) listed below.

________ (APPLICATION NUMBER) ________ (FILING DATE)

________ (APPLICATION NUMBER) ________ (FILING DATE)

COMBINED DECLARATION FOR PATENT APPLICATION AND POWER OF ATTORNEY (CONT'D) **(Includes Reference to Provisional and International (PCT) Applications)**	Attorney's Docket No. 032326-166

I hereby claim the benefit under Title 35, United States Code, § 120 of any United States applications(s) or International (PCT) Application(s) designating the United States of America that is/are listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in that/those prior application(s) in the manner provided by the first paragraph of Title 35, United States Code, § 112, I acknowledge the duty to disclose to the U.S. Patent and Trademark Office all information known to me to be material to the patentability as defined in Title 37, Code of Federal Regulations § 1.56, which became available between the filing date of the prior application(s) and the national or international filing date of this application:

PRIOR U.S. APPLICATIONS OR INTERNATIONAL (PCT) APPLICATIONS DESIGNATING THE U.S. FOR BENEFIT UNDER 35 U.S.C. § 120:

U.S. APPLICATIONS		STATUS *(check one)*		
U.S. APPLICATION NUMBER	U.S. FILING DATE	PATENTED	PENDING	ABANDONED
		☐	☐	☐

PCT APPLICATIONS DESIGNATING THE U.S.					
PCT APPLICATION NO.	PCT FILING DATE	U.S. APPLICATION NUMBERS ASSIGNED (if any)			
PCT/FR00/00150	January 24, 2000				

I hereby appoint the following attorneys and agent(s) to prosecute said application and to transact all business in the U.S. Patent and Trademark Office connected therewith and to file, prosecute and to transact all business in connection with international applications directed to said invention:

William L. Mathis	17,337	R. Danny Huntington	27,903	Gerald F. Swiss	30,113
Robert S. Swecker	19,885	Eric H. Weisblatt	30,505	Charles F. Wieland III	33,096
Platon N. Mandros	22,124	James W. Peterson	26,057	Bruce T. Wieder	33,815
Benton S. Duffett, Jr.	22,030	Teresa Stanek Rea	30,427	Todd R. Walters	34,040
Norman H. Stepno	22,716	Robert E. Krebs	25,885	Ronni S. Jillions	31,979
Ronald L. Grudziecki	24,970	William C. Rowland	30,888	Harold R. Brown III	36,341
Frederick G. Michaud, Jr.	26,003	T. Gene Dillahunty	25,423	Allen R. Baum	36,086
Alan E. Kopecki	25,813	Patrick C. Keane	32,858	Steven M. duBois	35,023
Regis E. Slutter	26,999	B. Jefferson Boggs, Jr.	32,344	Brian P. O'Shaughnessy	32,747
Samuel C. Miller, III	27,360	William H. Benz	25,952	Kenneth B. Leffler	36,075
Robert G. Mukai	28,531	Peter K. Skiff	31,917	Fred W. Hathaway	32,236
George A. Hovanec, Jr.	28,223	Richard J. McGrath	29,195		
James A. LaBarre	28,632	Matthew L. Schneider	32,814		
E. Joseph Gess	28,510	Michael G. Savage	32,596		

21839

and: ______________________________.

Address all correspondence to: James A. LaBarre
BURNS, DOANE, SWECKER & MATHIS, L.L.P.
P.O. Box 1404
Alexandria, Virginia 22313-1404

21839

Address all telephone calls to: James A. LaBarre ~~at (7~~03) 836-6620.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

COMBINED DECLARATION FOR PATENT APPLICATION AND POWER OF ATTORNEY (CONT'D) **(Includes Reference to Provisional and International (PCT) Applications)**	Attorney's Docket No. 032326-166

FULL NAME OF SOLE OR FIRST INVENTOR NACCACHE ~~David~~	SIGNATURE [illegible]	DATE 10 oct 2001
RESIDENCE (CITY & STATE/COUNTRY) 7, rue Chaptal, 75009, ~~PARIS,~~ FRANCE [illegible]		CITIZENSHIP FRANCE
POST OFFICE ADDRESS (HOME ADDRESS) 7, rue Chaptal, 75009 PARIS, FRANCE		
FULL NAME OF SECOND JOINT INVENTOR, IF ANY ROUSSEAU ~~Ludovic~~	SIGNATURE [illegible]	DATE 9 oct 2001
RESIDENCE (CITY & STATE/COUNTRY) Bât. A les Aires St Michel, 13400 ~~AUBAGNE~~, FRANCE [illegible]		CITIZENSHIP FRANCE
POST OFFICE ADDRESS (HOME ADDRESS Bât. A les Aires St Michel, 13400 AUBAGNE, FRANCE		
FULL NAME OF THIRD JOINT INVENTOR, IF ANY GIRARD ~~Pierre~~	SIGNATURE [illegible]	DATE 10 oct 2001
RESIDENCE (CITY & STATE/COUNTRY) 4 avenue Brue, 13600 LA CIOTAT, FRANCE [illegible]		CITIZENSHIP FRANCE
POST OFFICE ADDRESS (HOME ADDRESS 4 avenue Brue, 13600 LA CIOTAT, FRANCE		
FULL NAME OF FOURTH JOINT INVENTOR, IF ANY	SIGNATURE	DATE
RESIDENCE (CITY & STATE/COUNTRY)		CITIZENSHIP
POST OFFICE ADDRESS (HOME ADDRESS		
FULL NAME OF FIFTH JOINT INVENTOR, IF ANY	SIGNATURE	DATE
RESIDENCE (CITY & STATE/COUNTRY)		CITIZENSHIP
POST OFFICE ADDRESS (HOME ADDRESS		
FULL NAME OF SIXTH JOINT INVENTOR, IF ANY	SIGNATURE	DATE
RESIDENCE (CITY & STATE/COUNTRY)		CITIZENSHIP
POST OFFICE ADDRESS (HOME ADDRESS		
FULL NAME OF SEVENTH JOINT INVENTOR, IF ANY	SIGNATURE	DATE
RESIDENCE (CITY & STATE/COUNTRY)		CITIZENSHIP
POST OFFICE ADDRESS (HOME ADDRESS		
FULL NAME OF EIGHTH JOINT INVENTOR, IF ANY	SIGNATURE	DATE
RESIDENCE (CITY & STATE/COUNTRY)		CITIZENSHIP
POST OFFICE ADDRESS (HOME ADDRESS		
FULL NAME OF NINTH JOINT INVENTOR, IF ANY	SIGNATURE	DATE
RESIDENCE (CITY & STATE/COUNTRY)		CITIZENSHIP
POST OFFICE ADDRESS (HOME ADDRESS)		
FULL NAME OF TENTH JOINT INVENTOR, IF ANY	SIGNATURE	DATE
RESIDENCE (CITY & STATE/COUNTRY)		CITIZENSHIP
POST OFFICE ADDRESS (HOME ADDRESS		